

## PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2001-290657

(43)Date of publication of application : 19.10.2001

(51)Int.Cl. G06F 9/45

(21)Application number : 2000-111867 (71)Applicant : HITACHI LTD

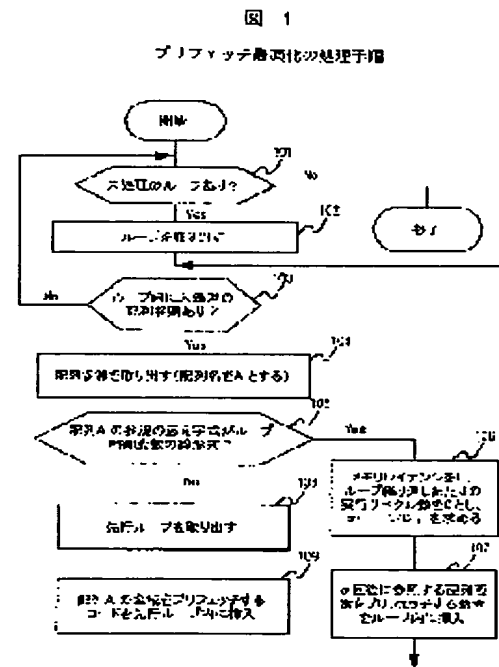
(22)Date of filing : 07.04.2000 (72)Inventor : MOTOKAWA KEIKO

### (54) SOFTWARE PREFETCH METHOD

**(57)Abstract:**

**PROBLEM TO BE SOLVED:** To provide a prefetch method for the reference of an array, in which a subscript expression is not a linear expression of a loop control variable.

**SOLUTION:** It is discriminated whether the reference of an array inside a loop is the subscript expression of the loop control variable (105) and when it is not a linear expression, a preceding loop which is to be executed preceding that loop is extracted (108). Then, an instruction for prefetching array data is inserted into the preceding loop and a code for prefetching all the area of the array in the preceding loop is generated (109).



## LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

**THIS PAGE BLANK (USPTO)**

Copyright (C); 1998,2003 Japan Patent Office

**THIS PAGE BLANK (USPTO)**

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2001-290657

(P2001-290657A)

(43) 公開日 平成13年10月19日 (2001. 10. 19)

(51) Int.Cl.<sup>7</sup>

G 0 6 F 9/45

識別記号

F I

C 0 6 F 9/44

データベース\* (参考)

3 2 2 C 5 B 0 8 1

審査請求 未請求 請求項の数 4 O L (全 7 頁)

(21) 出願番号 特願2000-111867 (P2000-111867)

(22) 出願日 平成12年4月7日 (2000. 4. 7)

(71) 出願人 000005108

株式会社日立製作所

東京都千代田区神田駿河台四丁目6番地

(72) 発明者 本川 敬子

神奈川県川崎市麻生区王禅寺1099番地 株

式会社日立製作所システム開発研究所内

(74) 代理人 100075096

弁理士 作田 康夫

Fターム(参考) 5B081 CC23

(54) 【発明の名称】 ソフトウェアプリフェッチ方法

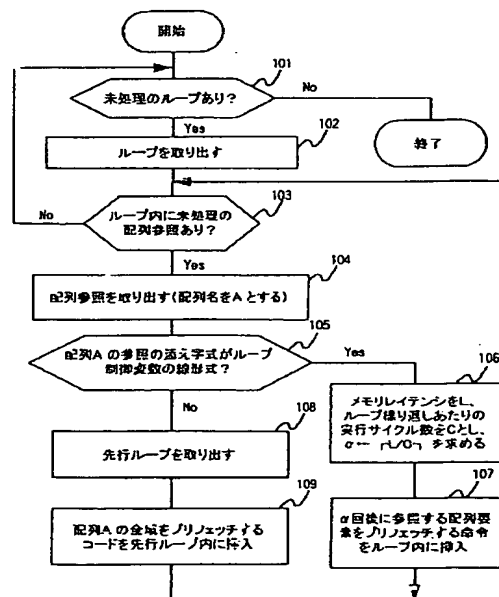
(57) 【要約】

【課題】 添え字式がループ制御変数の線形式でない配列参照に対するプリフェッチ方法を提供する。

【解決手段】 ループ内の配列参照がループ制御変数の添え字式かどうかを判定し (105)、線形式でない場合にはそのループよりも先行して実行される先行ループを取り出し (108)、先行ループ中に配列データをプリフェッチする命令を挿入して先行ループで配列全域をプリフェッチするコードを生成する (109)。

図 1

プリフェッチ最適化の処理手順



## 【特許請求の範囲】

【請求項1】 プリフェッチ命令を持つマイクロプロセッサに対する命令を生成するコンパイラにおいて、ループ中に含まれる配列参照に対するプリフェッチ命令を生成する方法であって、

該ループの実行よりも先行して実行される先行ループを取り出す、先行ループ取り出しステップと、先行ループで該配列の全域をプリフェッチするように、プリフェッチコードを先行ループ内に挿入する、全域プリフェッチコード生成ステップを有することを特徴とするソフトウェアプリフェッチ方法。

【請求項2】 請求項1のソフトウェアプリフェッチ方法であって、

配列参照の添え字式が配列参照を含むループのループ制御変数の線形式かどうかを判定するステップを有し、線形式でない場合には、前記先行ループ取り出しステップおよび、前記全域プリフェッチコード生成ステップを実行することを特徴とするソフトウェアプリフェッチ方法。

【請求項3】 請求項1または2のソフトウェアプリフェッチ方法を用いたコンパイラ。

【請求項4】 請求項1または2のソフトウェアプリフェッチ方法を用いたプログラムを格納した記憶媒体。

## 【発明の詳細な説明】

## 【0001】

【発明の属する技術分野】本発明は、計算機の利用技術において、オブジェクトプログラムの実行時間を削減するコンパイル方法に関する。特に、キャッシュへのプリフェッチ命令を挿入することによりキャッシュミス削減するソフトウェアプリフェッチ方法に関する。

## 【0002】

【従来の技術】マイクロプロセッサの速度向上に伴って主記憶アクセスのレイテンシが増大し、プログラムの実行性能に与える影響が増している。多くのプロセッサでは、主記憶よりもアクセスが速く比較的小容量のキャッシュ記憶を備え、レイテンシの大きい主記憶へのアクセス回数を削減している。すなわちメモリアクセス命令は、キャッシュヒット時には短いレイテンシでキャッシュをアクセスし、キャッシュミス時にのみ、主記憶をアクセスする。

【0003】キャッシュミス時の主記憶参照のレイテンシを隠す手法として、例えば「ToddC. Mowry他: Design and Evaluation of Compiler Algorithm for Prefetching, Architectural Support for Programming Languages and Operating Systems, pp.62-73, 1992」などに述べられているように、主記憶からキャッシュへデータを先行的に移動するプリフェッチ命令をプロセッサに用意し、コンパイラによってプログラム中にプリフェッチ命令を挿入する、プリフェッチ(ソフトウェア・プリフェッチ)と呼ばれる方法がある。プリフェッチ命令を利用

すれば、後続ループ繰り返して参照するデータを予め主記憶からキャッシュへ移動しておきながら、同時に別の演算を行うことができ、主記憶参照のレイテンシを隠すことができる。プリフェッチの適用対象は、添え字式がループ制御変数の線形式で表された配列参照である。

【0004】この方法では、例えば図7(a)に示すような、i番目のループ繰り返して配列要素a(i)を参照するようなループがあった場合、そのループの1回の繰り返しかかる実行サイクル数Cを見積もり、データをメモリからキャッシュに移動するのに要するサイクル数(メモリレイテンシ)LをCで割った値 $\alpha = \lceil L/C \rceil$

( $\lceil \cdot \rceil$ は小数点以下の切り上げを表す記号とする)を計算し、図8(b)に示すように、 $i + \alpha$ 回目の繰り返して参照するa(i+ $\alpha$ )をプリフェッチする命令をi回目の繰り返しに挿入する。このように $\alpha$ 回後の繰り返して参照するデータを予めプリフェッチしておくことにより、Lサイクル後にそのデータを参照するときにはデータが既にキャッシュに到着しているのでキャッシュヒットとなり、プログラム実行が高速化される。

## 【0005】

【発明が解決しようとする課題】従来のプリフェッチ方法では、参照データを含むループ内にプリフェッチコードを挿入する方法であり、プリフェッチ距離 $\alpha$ を求めて $\alpha$ 回後の繰り返して参照するアドレスに対するプリフェッチコードを出力する方法のため、添え字式がループ制御変数を含まないなど $\alpha$ 回後の参照アドレス式が不明の場合にはプリフェッチコードを生成することができなかった。このため、プリフェッチの対象は、添え字式がループ制御変数の線形式の配列参照に限られ、添え字式が線形式でない配列参照に対してプリフェッチを適用することができなかった。

【0006】プリフェッチされたデータが参照前にキャッシュから追い出されるのを防ぐためには、参照の直前にプリフェッチを完了させるために、 $\alpha$ 回前の繰り返してプリフェッチを発行するのが望ましい。プリフェッチをそれ以前に発行した場合にはキャッシュから追い出される可能性は高まるが、キャッシュ容量が参照されるデータ容量よりも十分大きければデータはキャッシュ上に残っている可能性が高く、プリフェッチの効果がある。

【0007】本発明の目的は、ループ内で参照される配列参照に対して、添え字式がループ制御変数の線形式でない配列参照に対してもプリフェッチによるメモリレイテンシ隠蔽の効果を得るプリフェッチ方法を提供することにある。

## 【0008】

【課題を解決するための手段】本発明の目的は、ループ中に含まれる配列参照に対して、その添え字式がループ制御変数の線形式かどうかを判定するステップと、線形式でないと判定されたときに、そのループよりも先行して実行される先行ループを取り出すステップ、および先

行ループ中にその配列のプリフェッチコードを挿入するプリフェッチ生成ステップにより達成される。

【0009】上記プリフェッチ生成ステップでは、先行ループの繰り返し全体で配列の全域をプリフェッチするようなプリフェッチコードを先行ループ中に挿入する。

【0010】

【発明の実施の形態】以下、図面を用いて本発明の実施の形態について説明する。

【0011】図2は、本発明によるコンパイラが稼動する計算機システムの構成図である。この計算機システムは、CPU201、ディスプレイ装置202、キーボード203、主記憶装置204、および外部記憶装置205より構成されている。キーボード203により、ユーザからのコンパイラ起動命令を受け付ける。コンパイラ終了メッセージやエラーメッセージは、ディスプレイ装置202に表示される。外部記憶装置205には、ソースプログラム206とオブジェクトプログラム207が格納される。主記憶装置204には、コンパイラ208、コンパイル過程で必要となる中間コード209およびループ表210が格納される。コンパイル処理はCPU201がコンパイラプログラム208を実行することにより行われる。

【0012】図3に、図2のシステムで稼動するコンパイラ208の処理手順を示す。

【0013】コンパイラの処理は、構文解析301、ループ解析302、プリフェッチ最適化303、レジスタ割り付け304、コード生成305の順で行う。

【0014】構文解析301では、ソースプログラム206を入力として構文解析を行い、中間コード209を出力する。構文解析処理に関しては、例えば「エイホ、セシィ、ウルマン著：コンパイラ（サイエンス社、1990年）」に記述されている。

【0015】ループ解析302により、プログラムに含まれるループの集合およびそれらのループの関係が求められ、これがループ表210に記録される。ループ解析についても、「エイホ、セシィ、ウルマン著：コンパイラ（サイエンス社、1990年）」に記述されている。

【0016】プリフェッチ最適化303は、本発明の特徴となるステップであるので、後で詳細に説明する。

【0017】レジスタ割り付け304では中間コード209の各ノードへのレジスタ割り付けを行う。コード生成305では、中間コード209をオブジェクトプログラム207に変換し、出力する。

【0018】図4はループ表210の例を示す図である。同図に示したループ表は、図8(a)のソースプログラムに対応するものである。

【0019】図4に示すように、ループ表210は、「ループ番号」欄401、「外側ループ」欄402、「直前ループ」欄403、「制御変数」欄404、「ループ回数」欄405、「実行サイクル」欄406から構

成されている。

【0020】「ループ番号」欄401には、各ループを特定するために、各ループに適宜割り当てられる番号が登録される。図4に示した例では、図8のプログラム中のコメントに示したように、4つのループにそれぞれ1から4までの番号が割り当てられている。

【0021】「外側ループ」欄402には、対応するループの外側のループのうち、最も内側のループのループ番号が登録される。図8の例では、ループ番号4のループの外側ループはループ番号3のループである。その他のループは外側ループを持たない。

【0022】「直前ループ」欄403には、同じ外側ループを持つループ同士（または外側ループを持たないループ同士）のうちで、直前に実行されるループのループ番号が登録される。図8の例では、外側ループが3のループはループ4だけなので、ループ4の直前ループはない。ループ番号1、2、3のループは外側ループが「なし」であり、これらの実行順を考えると、ループ3の直前ループはループ2、ループ2の直前ループはループ1である。ループ1は先頭ループであるので、直前ループはない。

【0023】「制御変数」欄404にはループ制御変数が登録される。図8の各ループのループ制御変数はそれぞれi, i, i, kである。

【0024】「実行サイクル」欄405には、ループ繰り返し1回あたりの実行サイクル数の見積もりが登録される。

【0025】次に、プリフェッチ最適化303の処理手順を詳細に説明する。

【0026】図1に、プリフェッチ最適化の処理手順を示す。各ステップを説明する。

【0027】ステップ101では、未処理のループがあるかどうかを判定し、あればステップ102で未処理ループを1つ取り出す。

【0028】ステップ103では、ループ内に未処理の配列参照があるかを判定し、あればステップ104で配列参照を1つ取り出し、この配列名をAとする。

【0029】ステップ105では、現在処理中の配列参照の添え字式を調べ、ループ制御変数の線形式かどうかを判定する。線形式ならば、ステップ106へ進む。

【0030】ステップ106では、メモリレイテンシ（プリフェッチに要するサイクル数）をLとする。Lはターゲットマシンによって決まっている値である。また、ループ繰り返しあたりの予測実行サイクル数をCとし、プリフェッチに必要なループ回数 $\alpha$ を $\lceil L/C \rceil$ により計算する。ステップ107では、 $\alpha$ 回後の繰り返しで参照する配列要素をプリフェッチする命令をループ内に挿入する。 $\alpha$ 回後の参照は、添え字式中のループ制御変数iを $i+\alpha$ に置き換えることにより生成する。

【0031】添え字式が線形式でないときには、ステッ

プ108へ進み、先行ループを取り出し、ステップ109で配列Aの全域をプリフェッチするコードを先行ループ内に挿入する。これらのステップは本発明の特徴となる部分である。

【0032】ステップ108の詳細な処理手順を図5に示す。

【0033】ステップ501で、処理対象のループをloopとし、ステップ502でloopの直前ループがあるかを判定する。直前ループがあれば、ステップ506へ進み、この直前ループを先行ループに決定して処理を終了する。

【0034】直前ループがなければステップ503へ進み、loopの外側ループがあるかどうかを調べる。外側ループがあればステップ504でloopに外側ループを設定し、ステップ502へ戻って、この外側ループの直前ループがあるかを調べる。このようにステップ502、503、504を繰り返し、直前ループが見つかるまで外側ループを辿っていく。

【0035】loopがプログラム先頭の最外側ループの場合には、直前ループも外側ループもないので、ステップ505へ進む。ステップ505では、プリフェッチ用にループ本体が空のループを作成し、loopの直前に挿入する。ループの挿入後には作成したループをループ表210に追加し、ループ表を更新する必要がある。尚、本ステップを省略し、先行ループが見つからない場合には、ステップ109のプリフェッチコード生成処理を中止してもよい。

【0036】ステップ109の詳細な処理手順を図6に示す。

【0037】ステップ601では、配列Aが何キャッシュライン分のサイズかを計算し、n\_linesに設定する。キャッシュラインサイズをLSバイト、配列AのサイズをSバイトとすると、n\_linesは「S/LS」により求められる。配列Aのサイズは、要素サイズ×要素数により求められるが、要素数が不明な場合などは、適当な値を仮定すればよい。

【0038】ステップ602では、ステップ108で取り出した先行ループのループ回数Nを調べ、n\_lines/NをPに設定する。Pは繰り返し1回あたりにプリフェッチすべきキャッシュライン数を表している。ループ回数Nが不明な場合には、適当な値を設定すればよい。また、繰り返しあたりのプリフェッチ命令数nにPを切り上げて整数にした値「P」を設定する。

【0039】ステップ603では、「offset=0」に相当するコードを生成し、先行ループのプリヘッダに挿入する。

【0040】ステップ604では、mを0で初期化する。

【0041】ステップ605ではmがn未満かどうか判定し、n未満ならばステップ606に進む。

【0042】ステップ606では「prefetch(&A+offset+LS\*m)」に相当するコードを生成して先行ループのループ本体中に挿入する。

【0043】ステップ607では、mに1を加算し、ステップ605に戻って、mがn未満の間ステップ606のプリフェッチ命令生成を繰り返す。ステップ605でmがn以上ならばステップ608へ進む。

【0044】ステップ608では、「offset=offset+LS\*P」に相当するコードを生成し、先行ループのループ本体の最後に挿入して、処理を終了する。

【0045】図6の処理手順では、先行ループの全繰り返しでプリフェッチ命令を実行するコードを生成しているが、先行ループ内の一部の繰り返しだけでプリフェッチを実施することも考えられる。例えば、先行ループの繰り返し回数が200、配列のキャッシュライン数が100ならば、先行ループの後半100回だけでプリフェッチを実行するような条件付きコードを挿入する方法も可能である。

【0046】以上でステップ303のプリフェッチ最適化の説明を終了する。

【0047】図8のプログラムを例に、本実施例の処理手順を説明する。

【0048】ステップ302でループ解析を行い、図4のループ表が作成される。

【0049】ステップ303のプリフェッチ最適化が図1の処理手順に従って実行される。

【0050】ステップ101で未処理のループを調べ、ステップ102でまずループ1を取り出す。ループ1はループ内にプリフェッチ対象となる配列参照がないため、ステップ103からステップ101に戻り、次にループ2の処理に進む。

【0051】ステップ104で配列参照「b(f())」を取り出す。ステップ105で添え字式「f()」が線形式でないと判定するので、ステップ108へ進み、図5に従い、先行ループを取り出す。ステップ502で、ループ2の直前ループはループ1であるので、ステップ506で先行ループにループ1を設定する。

【0052】次にステップ109へ進み、図6に従って処理を行う。ステップ601で、配列bのライン数を計算する。本例では、キャッシュラインサイズを32バイト、double型を8バイトとする。配列bのサイズは、8×400=3200バイトであるから、n\_lines=「3200/32」=100となる。

【0053】ステップ602では、先行ループであるループ1のループ回数が100であるから、P=100/100=1となり、繰り返しあたりのプリフェッチ命令数nは1である。

【0054】ステップ603でループ1の直前に「offset=0」を挿入する。

【0055】ステップ604でmを0で初期化し、ステ

ップ606で「prefetch(&b+offset1)」をループ1のループ本体中に挿入する。

【0056】ステップ607でmは1になり、ステップ605の判定後、ステップ608へ進んで、ループ1の本体の最後に「offset1=offset1+32」を挿入する。

【0057】ステップ103に戻り、ループ2には未処理の配列参照がないので、ステップ101に戻ってループ3の処理に進む。ループ3は配列参照を含まない（内側ループ内の配列参照は内側ループの処理対象とする）ので、次にループ4の処理に進む。

【0058】ステップ105で、ループ4の配列参照「c(g(i,k))」を取り出す。添え字式は線形式でないで、ステップ108へ進む。

【0059】ステップ502でループ4は直前ループを持たないので、ステップ504で外側ループであるループ3をloopに設定する。ループ3の直前ループはループ2なので、ステップ506で先行ループにループ2を設定する。

【0060】次にステップ109を実施する。ステップ601で、配列cのサイズは $8 \times 800 = 6400$ バイトであるので、ライン数は $\lceil 6400 / 32 \rceil = 200$ である。

【0061】ステップ602で、先行ループであるループ2のループ回数は100であるので、 $P=200/100=2$ となり、プリフェッチ命令数nは2である。

【0062】ステップ603でループ2の直前に「offset2=0」を挿入する。

【0063】nが2なので、ステップ605、606、607の処理は2度繰り返され、ステップ606で生成した「prefetch(&c+offset2)」および「prefetch(&c+offset2+32)」をループ2の本体中に挿入する。

【0064】最後にステップ608で「offset2=offset2+64」をループ2の本体の最後に挿入する。

【0065】

【発明の効果】本発明によれば、添え字式が線形式でない配列参照に対しても、先行ループ中で配列全域をプリフェッチすることにより、ループ開始時にデータがキャッシュ上にのっている可能性が高くなり、ループ中での配列参照時のキャッシュミス削減して、プログラムの実行を高速化する効果がある。

【図面の簡単な説明】

【図1】本発明に係るプリフェッチ方法におけるプリフェッチ最適化の処理手順を示す図である。

【図2】本発明に係るプリフェッチ方法を実施するコンパイラが稼動する計算機システムの構成図である。

【図3】本発明に係るプリフェッチ方法を実施するコンパイラの処理手順を示すフロー図である。

【図4】図8(a)のソースプログラムに対応するループ表の例を示す図である。

【図5】本発明に係るプリフェッチ方法における先行ループ取り出しの処理手順を示す図である。

【図6】本発明に係るプリフェッチ方法における配列全域プリフェッチコード生成の処理手順を示す図である。

【図7】従来技術によるプリフェッチの例を示す図である。

【図8】本発明によるプリフェッチ最適化の例を示す図である。

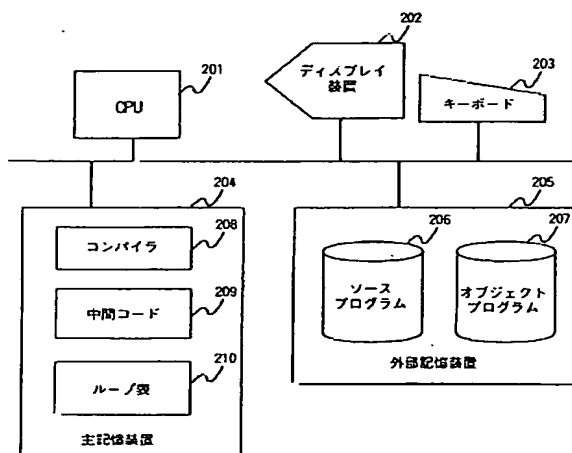
【符号の説明】

208…コンパイラ、303…プリフェッチ最適化処理、108…先行ループ取り出しステップ、109…配列全域プリフェッチコード生成ステップ。

【図2】

図 2

コンパイラが稼動するシステムの構成図



【図4】

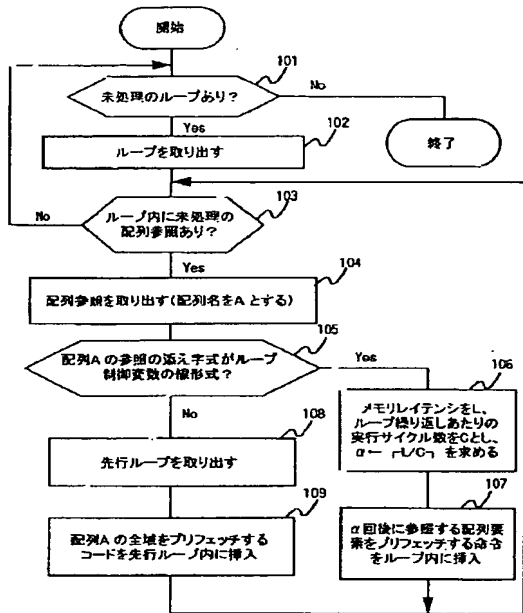
図 4

ループ表の例

ループ番号	外側ループ	直前ループ	制御変数	ループ回数	実行サイクル
1	なし	なし	i	100	2
2	なし	1	i	100	30
3	なし	2	i	100	500
4	3	なし	k	10	50

【図1】

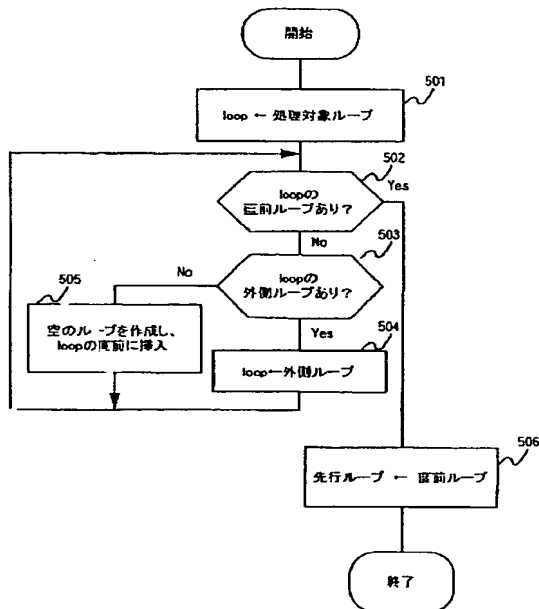
図 1  
プリフェッチ最適化の処理手順



【図5】

図 5

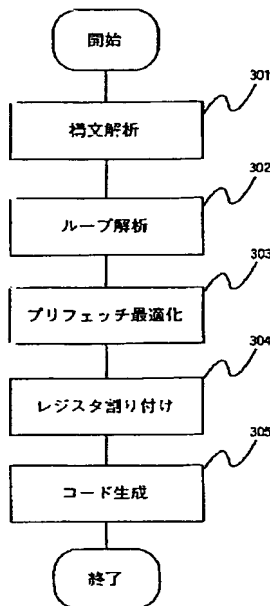
先行ループを取り出す処理手順



【図3】

図 3

コンパイラの処理手順



【図7】

図 7

従来方法によるプリフェッチ最適化の例

(a) ソースプログラム

```

for (i = 1; i <= 100; i++) {
    s = s + a[i];
}
    
```

(b) プリフェッチ最適化後のコード (ソースイメージ)

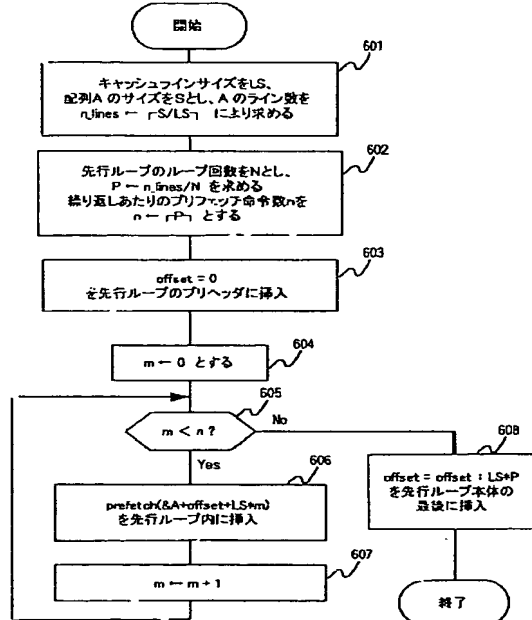
```

for (i = 1; i <= 100; i++) {
    s = s + a[i];
    prefetch(&a[i+α])
}
    
```

【図6】

図 6

配列Aの全域プリフェッチコードを生成する処理手順



【図8】

図 8

本発明によるプリフェッチ最適化の例

```
double b[400], c[800];
for (i = 0; i < 100; i++) {          /* loop1 */
    s0 = s0 + i;
}
for (i = 0; i < 100; i++) {          /* loop2 */
    s1 = s2 + b[f(i)];
}
for (i = 0; i < 100; i++) {          /* loop3 */
    for (k = 0; k < 10; k++) {        /* loop4 */
        s = s + c[g(i,k)];
    }
}
```

(a) ソースプログラム

```
double b[400], c[800];
offset1 = 0;
for (i = 0; i < 100; i++) {          /* loop1 */
    v[i] = i;
    prefetch (&b+offset1);          /* loop2のプリフェッチ */
    offset1 = offset1 + 32;
}
offset2 = 0;
for (i = 0; i < 100; i++) {          /* loop2 */
    a[i] = b[f(i)];
    prefetch (&b+offset2);           /* loop3のプリフェッチ */
    prefetch (&b+offset2+32);        /* loop3のプリフェッチ */
    offset2 = offset2 + 64;
}
for (i = 0; i < 100; i++) {          /* loop3 */
    for (k = 0; k < 10; k++) {        /* loop4 */
        s = s + c[g(i,k)];
    }
}
```

(b) プリフェッチ最適化後のコード (ソースイメージ)

**THIS PAGE BLANK (USPTO)**